VIDURA - AN INTERACTIVE MULTILINGUAL PUBLISHING SYSTEM - SPECIFICATION & DESIGN

SATYAJIT NATH, SUMANTA N. PATTANAIK and S.P. MUDUR

National Centre for Software Technology, Bombay, India

ABSTRACT

There are a large number of languages, many with their own distinct scripts, in current use in India. There are no real standards yet for Indian language text input, editing, etc. Word processing in Indian languages is yet to take off. Design of a publishing system for Indian languages requires specification right from text input, to text editing to page composition and printing. This paper presents the specification and design of an interactive multilingual publishing system for handling many of the Indian scripts including the Roman script. The system is being developed for the Department of Arts of the Government of India. Distinctive features and specific composition logic of the Indian scripts are briefly discussed first. Discussed next are the problems and inadequacies of extending commercially available desk-top publishing systems for handling context-sensitive scripts dominant in India. Then a unique keyboard input and text encoding scheme compatible with ASCII is presented followed by the publishing system specification with some design details. Unlike the available systems which are based on the 2D layout paradigm, throughout this system specification, the text editing paradigm is used to model the document and the user interface, even for the page make up task. This has also lead to a simple consistent user model for all the tasks of document input, editing and page composition, and also a single integrated representation for a document complete with layout attributes.

1. Introduction

Electronic Publishing has evolved starting with batch oriented text formatters to screen editors, word processors, turnkey photocomposing systems, sophisticated typesetting packages and now interactive publishing systems [Furuta82, Kernighan82, Knuth84, Mudur79, Reid80]. One of the hottest topics in computers is desk-top publishing and there is a plethora of new low cost systems claiming the ability to produce publication quality documents [Alsop87, Holmes87]. Desk-top publishing gives one complete control over the creation of a printed document. One can now key in text, edit text, draw illustrations, include photographs, design page layouts and also print a finished document with a relatively small and inexpensive computer and laser printer.

There are a few features which together distinguish desk-top publishing from the other electronic publishing tools such as wordprocessors,

typesetting packages, etc. [Seybold87]. Firstly, one can compose text in a manner that comes close to the requirements of typesetting. Secondly, the WYSIWYG bit mapped screen is supported and one can perform the composition task with interactive commands. Thirdly, they include a pagination or area composition program, letting one include graphics and output illustrative matter along with the text. They also provide reasonably good editing tools of the kind offered by the best word processors, to make textual changes interactively, not merely to uncomposed text, but to text in the process of interactive composition. Lastly, they support printing on a medium to high resolution raster printing device for text and graphics preferably interfacing via a page description language like PostScript[1]. As of today, very few commercially available desk-top publishing systems provide all these features.

In India, in current use, there are 16 different languages and hundreds of dialects with many of these languages having distinct scripts of their own. The English language, written in the Roman script, is also widely used, particularly for most official correspondence and higher education in science and technology. This has been largely due to the rapid advances that printing technology has made when dealing with the Roman script. Electronic publishing in Indian scripts is still in its infancy. When digital typesetting systems began to be imported for English language printing they evoked considerable amount of interest amongst academics from computer science as well as graphic arts.[Mudur80, Sinha84]. Considerable amount of preliminary work got done for adapting this technology to the Indian scripts. Typesetting and typefont design systems were indigenously developed. Even TEX and METAFONT [Knuth79] were used for experimental purposes with encouraging results [Ghosh83]. However, the commercially available systems are all based on traditions of the movable type and though very well suited to handle the Roman script are quite unsatisfactory and problematic when extended to handle the context sensitive scripts dominant in India [MacKay86]. For none of these scripts adapt themselves amicably to matrix bound character designs. What is really needed are electronic publishing systems specifically designed and developed to handle the Indian scripts and their specific graphic and composition requirements. Fortunately, most of the Indian scripts are derived from the Sanskrit script, Devanagari, and can be handled by the same logical mechanisms.

1 PostScript is a trademark of Adobe Systems Inc.

One important point to note here is that electronic publishing is really not a new concept for the Roman script. Established standards exist for handling of text in this script including editors and wordprocessors. Available publishing systems are designed to and very often rely heavily on being able to incorporate text created by other systems. The centre piece of these publishing systems is thus the pagination or area composition software. On the other hand, electronic publishing of Indian language texts has yet to take off. There are no standards yet for keyboard input, text encoding, text editing, etc. Word processing programs available are very few and usually quite dismal in performance. There do exist manual typewriters. Their design is based more on graphical primitives and less on the known alphabets of the languages. Composing graphic primitives into letter shapes is the burden of the typist. It is well understood that such a layout is unsuited for computer input mechanisms [DoE86]. One immediate implication of this is that whereas the designer of a publishing system for the Roman script has to concentrate only on the page make-up task, for Indian languages the designer has to worry about specifications for keyboard input mechanisms, to text editing to page composition and printing. There is however one positive aspect to this entire design task, which is that it is now possible with some care to base the system design on efficient and consistent models for document maintenance and user interface. This, in our opinion, is something that the available Roman script publishing systems have been unable to do. The rest of this paper describes the specification and some design details of an interactive publishing system, Vidura, specifically designed to handle Indian scripts including the Roman script.

2. Distinctive Features of Indian Scripts

In the Roman script, as is used for English, the phonetic form of words in the language, basically pronunciation, is ambiguous (the but-put problem) i.e., there is no exact match between phonemes and orthographic representations. This information is based on arbitrary historical rules and is memorized by people using the language. Other European languages add some diacritical marks to represent this information. Even then, ambiguity is not totally eliminated. However, due to this fortuitous reason, in the Roman script, there is no Context-Sensitive composition, except for adding diacritical marks. Unmodified characters from the base alphabet are always added linearly to the right. Thus, composition of syllables and consequently of words is strictly a process of juxtaposing alphabet symbols in a

horizontal direction. In Devanagari and related Indian languages, a phoneme is directly represented in a unique graphical form. The basic graphical unit in these Indian scripts is the AKSHARA. Aksharas are added linearly to form words and words are similarly added linearly to form lines. In this respect, composition rules in Devanagari and Roman are identical. But, the most important difference lies in the rules governing the graphic form of Aksharas. Aksharas are formed out of a basic alphabet consisting of 34 CONSONANTS (C), 2 CONSONANT MODIFIERS (Cm), 14 VOWELS (V) and 3 VOWEL MODIFIERS (Vm). Some South Indian languages have more than 14 vowels.

An Akshara could be of two types:

- a sequence of one or more consonants, each optionally followed by a consonant modifier, followed by a vowel, optionally followed by a vowel modifier, i.e. C [Cm] [C [Cm] ... C [Cm]] V [Vm].
- a lone vowel, optionally followed by a vowel modifier, i.e. V [Vm].

Figure 1: *the basic alphabet set and various graphical forms*

Standalone Vowels : अ आ इ ई उ ऊ ऋ ए ऐ ओ औ

Matra forms : ा ि ी ु ू ृ े ै ो ौ

Extended Vowels : अॅ अॅ अॅ ओ ओ ओ इ उ ऊ

Matra forms : ॅ ॅ ॅ ॅ ॅ ॅ ि ु ू

Matra Formation of Vowels : क् + ई = की { Matra of ई }

इ + उ = इु { Matra of उ }

Examples of Consonants :

Half-form : क श ड् ह फ ब

Full-form : क श ड ह फ ब

Examples of Conjuncts :

Half-form : क्ष { क् + ष् }, श्र { श् + र् }

Full-form : क्ष { क् + ष् + अ }, श्र { श् + र् + अ }

The consonant र as diacritical marks in conjuncts :

क्र (क् + र्), ट्र (ट् + र् + अ), प्र (प् + र्),

र्क (र + क), र्ठ (र + ठ), र्प (र + प)

The sequence of consonants in a Type-I Akshara results in different graphic forms depending on the consonants involved. Some consonants combine to form completely new graphic forms called CONJUNCTs (Samyuktaksharas). Other consonants combine linearly as a simple juxtaposition of their HALF-FORMs.[1] In any given Type-I Akshara, one or more sub-sequences of consonants may result in conjuncts. Half-forms and conjuncts combine linearly and are almost always restricted to the X-height region.

The graphical form of a lone vowel in a Type-II Akshara is absolutely different from that of the vowel in a Type-I Akshara which has consonants immediately preceding it. In the latter case, the vowel takes the form of a MATRA (a diacritical mark) which attaches to the consonant; whereas in the former a stand-alone graphic form results. Unlike in the Roman script where diacritical marks usually appear in the ascender or descender region of a letter, MATRAs in Indian scripts and vowel/consonant modifiers may appear even in the X-height region of a letter. In fact, some MATRAs may occur in all regions and even to the left of the preceding consonant sequence. Figure 1 shows the basic alphabet set and the various graphical forms. A one-to-one correspondence between an alphabetic character and a type cannot be assumed. This is true also for diacritical marks on Roman letters. But whereas this is an exception in the Roman script, for Indian scripts it is the general rule. As pointed out earlier, it is extremely difficult to work with composition logic that is based on matrix bound character graphics unless an extremely large graphic character set is maintained. This, however, introduces its own incumbent problems of keyboarding and encoding because it is not possible to restrict the set to say, within 256 or so. Kern tables, accent marks and ligatures - some of the mechanisms available in today's systems - are being used for handling Indian scripts. The resulting solutions are not only cumbersome, inefficient and limited in many ways but also result often in graphically unsatisfactory output.

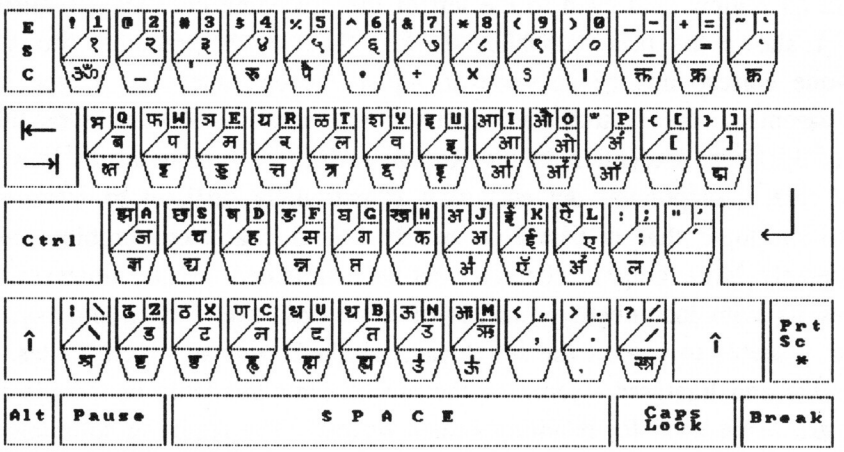
3. The Input Mechanism and Character Encoding

As suggested earlier, the phonetic division of Indian alphabets into vowels, consonants and their modifiers serves as a common base for all Indian scripts. The methodology of combining these basic groups to form syllables is governed by a set of well-defined rules, laid down during the time of Panini, the great grammarian and linguist. The basic principle is to com-

1 Note: In fact this is the only case in which Akshara composition has rules similar to those of Roman.

pletely distinguish between vowel sounds and consonant sounds. Therefore, to vocalize a syllable, one either pronounces a pure vowel sound or a sequence of pure consonant sounds followed by a vowel sound. The consonant modifiers serve to modify the consonant sounds and the vowel modifiers do the same for vowels. On this basis we have described aksharas as Type-I or Type-II. Hence, given a sequence of characters denoting pure consonants and vowels we can unambiguously compute their partitioning into a sequence of Type-I and Type-II aksharas by going through the sequence one at a time from start to end. Figure 2 shows the PC keyboard layout. We have taken great care to ensure that we only replace the Roman script key positions of the QWERTY keyboard. For ease of use we have also provided frequently used conjuncts and very infrequently used vowels and modifiers in the ALT position of the keyboard. For a detailed analysis of this approach, see [Pattanaik87].

Figure 2: *the PC keyboard layout*



The phonetic scheme used for inputting from the keyboard is also used for representing text in the computer. Although most Indian languages follow the generic rules of composition of Aksharas described above, the graphical forms for aksharas as well as composition rules for individual MATRAs and modifiers vary from one language to another. Thus, to cite an example, a vowel MATRA which attaches in the super level of the Akshara in Devanagari might attach in the ascender X-height level in Assamese (ae-kaar). Such variations show that though Aksharas, in principle, are formed

exactly in the same manner in all these languages, the final graphic form may be widely varying. One can therefore, think of a common representation which will encompass all these languages and be independent of the final graphic form of the aksharas.

There are two aspects to the task of assigning codes to the basic script primitives. Since implementation is being carried out on a machine which uses ASCII for the Roman script, it is essential that no damage be done to this system while adding our codes. Secondly, the codes must be assigned in such a way that the dictionary sorting order for Devanagari is maintained. Figure 3 shows the encoding for Devanagari. The same encoding applies to all Indian scripts. For multiple languages we use nested SI and SO codes with a single additional byte to denote the language. Note that the 7 bit ASCII codes for Roman have been left untouched.

Figure 3: *encoding for Devanagari*

L/H	2	3	4	5	6	7	10	11	12	13	14	15
0:	SP	0	@	P	`	p	SP	०	@	औ	'	भ्
1:	!	1	A	Q	a	q	!	१	9	.	झ्	म्
2:	"	2	B	R	b	r	"	२	S	..	ञ्	य्
3:	#	3	C	S	c	s	#	३	अँ	क्	द्	र्
4:	$	4	D	T	d	t	$	४	अं	ख्	ठ्	ल्
5:	%	5	E	U	e	u	%	५	अः	ग्	ड्	ळ्
6:	&	6	F	V	f	v	&	६	अ	घ्	ढ्	व्
7:	'	7	G	W	g	w	'	७	आ	ङ्	ण्	श्
8:	(	8	H	X	h	x	(	८	इ	च्	त्	ष्
9:	)	9	I	Y	i	y	)	९	ई	छ्	थ्	स्
10:	*	:	J	Z	j	z	*	:	उ	ज्	द्	ह्
11:	+	;	K	[	k	{	+	;	ऊ	[	ध्	{
12:	,	<	L	\	l	|	,	<	ऋ	\	न्	|
13:	-	=	M	]	m	}	-	=	ए	]	प्	}
14:	.	>	N	^	n	~	.	>	ऐ	^	फ्	~
15:	/	?	O	_	o	DEL	/	?	ओ	_	ब्	DEL

As mentioned earlier, an akshara could be a sequence of one or more characters. For interactive display purposes, intermediate graphic forms of aksharas have to be displayed. With every new character appended to or deleted from (during editing) an akshara, the graphic form displayed has to be updated. The akshara structure can be elegantly represented by a tree. Each edge in a tree represents a coded primitive and each node represents a

graphic symbol to be displayed [Mudur80]. Thus any Indian language can be regarded as a set of akshara trees [1]. Therefore, in the simplest case, an algorithm for displaying aksharas will "walk through" appropriate trees in the set by following edges governed by the code sequence and displaying graphic forms whenever a leaf node is reached. In the case of interactive display, the algorithm will display the graphic form for each node at every level.

Conceptually, this scheme is ideal for the task. However, from an implementation point of view, it requires a lot of storage. This happens because, in many cases, there are simple rules of graphic composition for a group of aksharas and this information is not utilized in this scheme. For example, addition of a vowel to a consonant causes the pure consonant form to change to the full form with the vowel MATRA getting attached to it. Similarly, addition of a consonant to a consonant may cause either the pure consonant changing to the half form and the new consonant taking the pure form, or the pure consonant changing to a pure conjunct form. In the tree scheme, for each akshara, with one consonant and one vowel, there would be a separate tree, i.e., a total of 36 x 12 trees. An alternative solution is to classify various types of aksharas and have separate procedures for displaying them. This scheme has lower memory requirements but is computationally expensive. Depending upon what is more important, speed or storage, a choice has to be made between the two schemes. In an interactive system, real-time display is very important. Hence we have adopted the former by flattening out the set of trees into a 2D table accessed by a pair of consecutive primitive codes.

4. The Usage Paradigm For Vidura

With emphasis on bit mapped multi-window graphics, pop-up or drop-down menus, icon driven interfaces and mouse-based pointing and positioning, the two dimensional layout paradigm as used for engineering draughting, interior design layout, etc., seemed natural and fashionable to choose as a paradigm for page composition. And this is what all systems have done. It has a lot of high-tech appeal too for the lay user. However, we do not at all believe that this is the ideal paradigm for publishing. In fact, on rectangular pages, horizontal and vertical alignments are crucial both aesthetics as well as readability wise. Hence, arbitrary dragging of paragraphs within a

1 A matter of interest is that the Roman script is a special case in this structure - it is a set of trees with only one node in each tree.

page and snapping to grids are far too general, unconstrained and somewhat unnatural operations for page composition or column setting. For example, without elaborate geometric interference checks, one cannot ensure that matter is being set with proper placement and no unintended superimposition. Another undesired side effect of this is that most publishing systems do not focus at all on the creation of the matter (text and graphics) in the document. Some limited capabilities are included without properly integrating these into the system, for it is assumed that text and graphic elements will be imported during the time of composition. As a result, most of the systems do not properly handle large publications, cannot reflect edit changes or perform global edits or do h & j etc., essentially tasks which are handled elegantly and efficiently by many screen editors and word processors.

Considering the fact that more than a substantive part of the total task of publishing a document happens to be creation and editing of the matter that goes in, and additionally, the earlier observation, that electronic document creation tools have only now begun to emerge for Indian scripts, we decided when working on Vidura that we shall base the specification on the text editing paradigm. For, this paradigm is simple and easy to understand. Also formal specifications have been worked upon [Sufrin82]. Below we shall informally elaborate on this paradigm before discussing in some detail the design specification and how this paradigm has had its influence.

In the text editing paradigm, typically we have edit and display mechanisms.

- Editing is always described around a current point of intereset, the cursor. Editing operations constitute moves, inserts, replaces and deletes. All these operations are defined over a hierarchical structure of syntactic constructs from the cursor point. Typical syntactic constructs in editing are: characters, words, lines, paragraphs, pages, blocks and the document itself.

- The display mechanism can be likened to one that enables looking at the document through a movable window which locates itself so as to keep the cursor in view. The matter within a window is projected into a two-dimensional region with typographic attributes taking effect wherever applicable.

5. Vidura Specifications

Vidura specifications are entirely based on the one single paradigm of text editing.[1] This has resulted in a consistent, simple, elegant and powerful user interface and an integrated document structure. Four key design concepts that have facilitated this are - (a) a notional composite cursor with typographic attributes, (b) nested typographic and composition attributes using a block structured scope rule, (c) A hierarchical box structure for generating composed pages, and (d) composition style macros.

The document is modelled simply as a sequence of multi-lingual letters or glyphs (Indian script aksharas, Roman script letters, ASCII symbols, etc.), with composition and typographic control attributes associated with some subsequences and with some positions (between two glyphs) in the sequence. At any instant of time there is a single point of interest, i.e., the cursor. It always lies between two glyphs. The cursor is always associated with a complete setting of typographic attributes. Attribute settings are scoped and nested (block structured) [Mudur79]. That is, they apply over some contiguous subsequence of the document. There are three ways in which attribute settings get associated with the cursor. One, when the cursor is explicitly moved into or out of an attribute scope. Another, when a new attribute setting is inserted at the current cursor position. This begins a new attribute scope which has the cursor within it but nothing else. As letters are inserted they take on the typographic attributes of the cursor and become part of the current scope. And a third, when there is deletion of text that results in an attribute setting with empty scope (no text or the cursor) or when there is no text separating nested settings. The cursor can be moved over syntactic units in the document. The cursor may lie anywhere within a syntactic unit other than an akshara or letter. Standard move and delete operations, as in any other wordprocessor, are available. Because of the aksharas being distinct glyphs, a special DELETE key is needed for intra-akshara editing.

The display mechanism is the one which outputs the matter in the document in the form of composed pages. A subtle but fundamentally important difference from many of the available systems must be noted here. The basic idea is to have one or more movable windows defined over the document and project these suitably onto the regions in a page as against the approach in existing systems of importing text or other matter into a region

1 In fact we have also used this paradigm for drawing and editing of illustrations; but that we shall be describing elsewhere.

in a page. The parameters of the projection define a composition style. It is clear that in this mechanism one can support predefined composition styles, a very useful and powerful facility that is commonly available in large type setting packages operating in batch mode and in very few interactive publishing packages [Holmes87, Mudur82].

A document is displayed as a sequence of non-overlapping pages. A page is the largest possible box. It may be tessellated into smaller non-overlapping boxes which may also be tessellated into smaller non-overlapping boxes. The document display structure is therefore hierarchical. Pages are identified by a page number while each box within could have a unique name - *header* , *footer, footnote, caption,* etc . Except the page, all boxes have a type: text, image or empty. Empty boxes are white space in the page. A box has a default attribute setting associated which is applicable to matter that is set into the box. All child boxes inherit attributes from their parent unless they are explicitly altered. The text box attributes are classified into types like dimensions, stretchability (size of a box depends on its contents), matter permanency (useful for running heads, etc.), layout parameters and typographic parameters. For detailed design specifications and user interface description, see [Nath87].

6. Implementation Status

The Vidura multilingual publishing system is being developed to operate on an IBM PC/AT with a medium resolution graphics screen and supporting at present PostScript printers, many dot matrix printers (for draft copies) and the DEC LN03 laser printers. Initially the Roman and Devanagari scripts will be incorporated. A PostScript interpreter has been developed and supports dot-matrix printers and the DEC LN03 laser printer. Digital typeface design too is in progress. A prototype of the Vidura system is expected to be demonstrated towards the end of 1987.

Acknowledgements

We are extremely grateful to Prof. R.K. Joshi, IIT, Bombay, for his involvement in all the linguistic aspects of this work, particularly the keyboard layout and the typeface design. His enthusiasm as well as his own love, his Deshascript, a common script for all Indian languages, have been strong motivating forces. We thank the Indira Gandhi National Centre for Arts for supporting the total development of Vidura. We are also grateful to Dr. S. Ramani, Director, National Centre for Software Technology, for his sup-

port and encouragement. We are grateful to Shri V. Padmanabhan for all his efforts in ensuring that this manuscript gets produced in time.

References

[1] Alsop Stewart, DeskTop Publishing Without Hype, *PC MAGAZINE*, 111-156, Feb.10, 1987.

[2] Dept. of Electronics, Govt. of India, Report of the Committee for Standardization of Keyboard Layout for Indian Script based Computers, *Electronics - Information & Planning*, **14**, 1, Oct. 1986, 3-18.

[3] Furuta Richard, Jeffrey Scofield and Alan Shaw, Document Formatting Systems: Survey, Concepts, and Issues. In *DOCUMENT PREPARATION SYSTEMS*, eds. J.Nievergelt et al, North Holland Publishing Company, 1982.

[4] Ghosh P.K., *An Approach to Type Design And Text Composition in Indian Scripts*, Report No. STAN-CS-83-965, Stanford University, April 1983.

[5] Holmes Thom, Make My Page!, *BYTE*, **12**, 5, 159-166.

[6] Kernighan Brian W., Michael E. Lesk, UNIX Document Preparation. In *DOCUMENT PREPARATION SYSTEMS*, eds. J. Nievergelt et al, North Holland Publishing Company, 1982.

[7] Knuth D.E., *TEX and METAFONT: New Directions in Typesetting*, Digital Press, 1979.

[8] Knuth D.E., *The TEX Book*, Reading, MA: Addison-Wesley, 1984.

[9] MacKay Pierre A., Typesetting Problem Scripts, *BYTE*, **11**, 2, 201-218.

[10] Mudur S.P., A.W. Narwekar, Abha Moitra, Design of Software for Text Composition, *Software - Practice and Experience*, **9**, 313-323.

[11] Mudur S.P., R. Sujatha, Three Systems for Typesetting - A Survey, *Computer Science and Informatics*, **12**, 1, 3-9.

[12] Mudur S.P., L.S. Wankankar, P.K. Ghosh, *Design Information Report on Text Composition in Devanagari, RIND Report*, Research Institute for Newspaper Development, Madras, Sep. 1980.

[13] Nath Satyajit, S.P. Mudur, *Vidura Functional Specifications and User Interface*, Internal Design Report, Graphics & CAD Division, NCST, 1987.

[14] Pattanaik Sumanta N., Satyajit Nath, S.P. Mudur, Computer Processing of Indian Scripts - A Pure Consonant Approach, In *Proc. of Computer Assisted Language Processing Seminar*, Delhi, Sep. 1987.

[15] Reid Brian, *SCRIBE - A Document Specification Language and its Compiler*, Ph.D. Thesis, Dept. of Computer Science, Carnegie Mellon University, 1980.

[16] Seybold John W., The Desktop-Publishing Phenomenon, *BYTE*, **12**, 5, 149-154.

[17] Sinha R.M.K., Computer Processing of Indian Languages and Scripts - Potentialities and Problems, *Journal of Institute of Electronics & Telecommunication Engineers (India)*, **30**, 6.

[18] Sufrin Bernard, Formal Specification of a Display - Oriented Text Editor, *Science of Computer Programming*, 1982, 1, 157-202.